An optimal $O(\log \log n)$ time parallel string matching algorithm [1]

Dany Breslauer
Columbia University

Zvi Galil
Columbia University
and Tel-Aviv University

March 1989

CUCS 492-89

[1] Work supported byNSF Grants CCR-86-05353 and CCR-88-14977

Abstract

An optimal $O(\log\log n)^*$ time parallel algorithm for string matching on CRCW-PRAM is presented. It improves previous results of [G] and [V].

*All logarithms are to the base 2

1 Introduction

On a CRCW-PRAM we can solve some problems in less than the logarithmic time needed on weaker models such as CREW-PRAM. For example *OR* and *AND* of n input variables, and finding the minimum or maximum of integers between 1 and n (see section 7) can be done in $O(1)$ time using n processors. Finding the maximum in the general case takes $O(\log\log n)$ time on $n/\log\log n$ processors ([Va] and [SV]), and the same is true for merging ([Va], [Kr] and [BH]). Recently, few more $O(\log\log n)$ optimal parallel algorithms have been found for finding prefix minima [Sc], all nearest neighbors in convex polygons [ScV], triangulation of a monotone polygone and finding nearest smallers [BSV]. We show that the string matching problem can be solved in $O(\log\log n)$ time with $n/\log\log n$ processors too, establishing that it belongs to one of the lowest parallel complexity classes.

The problem of string matching is defined as follows: Given two input arrays $TEXT(1\cdots n)$ and $PATTERN(1\cdots m)$, find all occurrences of the pattern in the text. Namely, find all indices j such that $TEXT(j+i-1) = PATTERN(i)$, for $i = 1\cdots m$. In the sequential case, the problem can be solved using the two well known linear time algorithms of Knuth, Morris and Pratt [KMP] and Boyer and Moore [BM]. In the parallel case, an optimal algorithm discovered by Galil [G] for fixed alphabet and later improved by Vishkin [V] for general alphabet solves the problem in $O(\log n)$ time on a CRCW-PRAM. Recall, that an *optimal* parallel algorithm is one with a linear time-processor product. We use the weakest version of CRCW-PRAM: the only write conflict allowed is that processors can write the value 1 simultaneously into a memory location.

Our algorithm solves the string matching problem for general alphabet in $O(\log\log m)$ time using $n/\log\log m$ processors on a common CRCW-PRAM. It is based on the previous two optimal algorithms, and similarly works in two stages. In the first, we gather some information about the pattern and use it in the second stage to find all the occurrences of the pattern in the text. The output of the algorithm is a Boolean array $MATCH(1\cdots n)$ which has the value 'match' in each position where the pattern occurs and 'unmatch' otherwise.

Suppose we have mn processors on a CRCW-PRAM, then we can solve the string matching problem in $O(1)$ time using the following method:

- First, mark all possible occurrences of the pattern as 'match'.
- To each such possible beginning of the pattern, assign m processors. Each processor compares one symbol of the pattern with the corresponding symbol of the text. If a mismatch is encountered, it marks the appropriate beginning as 'unmatch'.

Assuming we can eliminate some of the possible occurrences and have only l left (ignoring the problem of assigning the processors to their tasks), we can use the method described above to get an $O(1)$ parallel algorithm with lm processors. Both [G] and [V] use this approach. The only problem is that one can have many occurrences of the pattern in the text, even much more than the n/m needed for optimality in the discussion above.

To eliminate this problem, we use the notion of the period suggested in [G] and also used in [V]. A string u is called a *period* of a string w if w is a prefix of u^k for some positive integer k or equivalently if w is a prefix of uw. We call the shortest period of a string w *the period* of w.

Suppose u is the period of the pattern w. As explained below, we cannot have two occurrences of the pattern at positions i and j of the text for $|j - i| < |u|$. If instead of matching the whole pattern, we look only for occurrences of u, assuming we could eliminate many of them and have only $n/|u|$ possible occurrences left, we can use the $O(1)$ algorithm described above to verify them using only n processors. Then by counting the number of consecutive matches of u, we can match the whole pattern.

In many cases, we slow down some computations to fit in our processor bounds. This is done using a theorem of Brent [B], which allows us to count only the number of operations performed without concern about their timing.

Theorem (Brent). Any synchronous parallel algorithm of time t that consists of a total of x elementary operations can be implemented on p processors in $\lceil x/p \rceil + t$ time.

Using this theorem for example, we can slow down the $O(1)$ time string matching algorithm described above to run in $O(s)$ time on lm/s processors.

Brent's Theorem as well as other computations described below require the assignment of processors to their tasks which in our case is done using standard techniques.

In section 2 we review two facts on periods from [G] and in section 3 we review the notion of witness from [V]. In sections 4–6 we describe the algorithm. Section 7 is devoted to some technicalities left out in the previous sections.

2 Periodicity properties

We will use some simple facts about periods in the next sections. The proof can be found in [G].

1. If w has two periods of length p and q and $|w| \geq p + q$, then w has a period of length $\gcd(p, q)$ ([LS]).

2. If w occurs in positions p and q of some string and $0 < q - p < |w|$ then w has a period of length $q - p$. Therefore we cannot have two occurrences of the pattern at positions p and q if $0 < q - p < |u|$ and u is the period of the pattern.

3 Witnesses

An important idea in our algorithm is a method suggested in [V], which enables us to eliminate many possible occurrences in $O(1)$ time. One computes some information about the pattern which is called $WITNESS(1 \cdots m)$ in [V], and uses it in the second stage for the analysis of the text.

Let u be the period of the pattern w, and let v be a prefix of w. It follows immediately from the periodicity properties that if $|u|$ does not divide $|v|$ and $|v| < \max(|u|, |w| - |u|)$, then w is not a prefix of vw. In that case we can find an index k such that

$$PATTERN(k) \neq PATTERN(k - |v|).$$

We call **this k a witness** to the mismatch of w and vw, and define

$$WITNESS(|v| + 1) = k.$$

We are interested only in $WITNESS(i)$ for $1 < i \leq |u|$ which by fact 2 can be based only on the first $2|u| - 1$ symbols of the pattern. Suppose we already computed $WITNESS(i) \geq 2|u|$, let $r = WITNESS(i) \ mod \ |u|$, then, if $r < i$, we set $WITNESS(i)$ to $r+|u|$, otherwise we set $WITNESS(i)$ to r.

4 Duels and Counting

Assume that u is the period of the pattern w, $w = u^k v$, v is a proper prefix (possible empty) of u and $p = |u|$. We call the pattern *periodic* if its length is at least twice its period length (i.e. $m \geq 2p$). Having computed the $WITNESS$ array in the first stage, Vishkin [V] suggests the following method to eliminate close possible occurrences which he calls a *duel*. Suppose we suspect that the pattern may start at positions i and j of the text where $0 < j - i < |u|$, thus, since we computed $r = WITNESS(j - i + 1)$ we can find in $O(1)$ time a symbol in the text which will eliminate one or both of the possible occurrences. More specifically, since $PATTERN(r) \neq PATTERN(r - j + i)$, at most one of them can be equal to $TEXT(r + i - 1)$ (see figure 1).

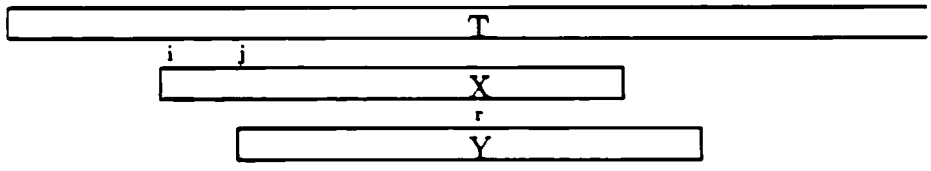

Figure 1. $X \neq Y$ and therefore we cannot have $T = X$ and $T = Y$.

Actually, we eliminate possible occurrences of some prefix of the pat-

tern. In the periodic case, we saw in the previous section that the witness information can be based only on the first $2p$ symbols of the pattern, thus we eliminate positions in which there is no occurrence of u^2. While in the nonperiodic case, the witness information is based on the whole pattern and positions where there is no occurrence of it can be eliminated. Having many such duels in pairs, the algorithm of [V] eliminates enough possible occurrences of u in the text in $O(\log m)$ time and verifies them using the $O(1)$ time algorithm described above. We manage to reduce the time of [V] to $O(\log\log m)$ time algorithm using the following observations:

- Duels "work like" maximum. Having a block of the text of length equal to p, only one occurrence of the pattern might start in it. Assume that the pattern can start anywhere within that block, and suppose we have p^2 processors. Assign a processor to each pair and perform a duel. Since in every pair at least one loses, at the end we are left with no more than one possible occurrence in each block. The exact details of the algorithm appear in the next sections.

- We simplify the "counting" of consecutive occurrences of u in the text in the periodic case. A recent result of Beame and Hastad [BHa] shows that computing the parity of n bits on a CRCW-PRAM takes $\frac{\log n}{\log\log n}$ with any polynomial number of processors, so no "real" coutings is possible within our time bounds. Assume without loss of generality that the text is of length $n = 2m-p$ (divide the text into $\frac{n}{m-p} = O(\frac{n}{m})$ overlaping groups of length $2m - p$). We call an occurrence of u^2 at position i an *initial occurrence* if there is no occurrence of it at position $i - p$. We call such occurrence a *final occurrence* if there is

no occurrence at position $i+p$. The main observation is that there is at most one initial occurrence of interest which is the rightmost initial occurrence in the first $m-p$ positions. Any initial occurrence in a position greater than $m-p$ is of no interest since there are not enough symbols in the text to match the whole pattern. Since the pattern is periodic with period length p initial occurrences which are smaller cannot start occurrences of the pattern either. The corresponding final occurrence is the smallest final occurrence which is greater than the initial occurrence.

5 Processing the text

As we mentioned above, duels are like maximum. We describe an optimal $O(\log\log m)$ time text analysis based on having $WITNESS(2\cdots r)$, for $r=\min(p,\lceil m/2\rceil)$ computed in the pattern analysis stage that works similarly to the maximum finding algorithm of [SV]. Recall that $p=|u|$ is the length of the period of the pattern. In the periodic case we divide the text into groups of length $n=2m-p$, while in the nonperiodic case we work on the whole text.

We have $WITNESS(i)<2p$. Partition the text into blocks of length r. We have n/r such blocks. In each block mark all positions as possible occurrences. Partition them into groups of size $\sqrt{r}$ and repeat recursively. The recursion bottoms out with one prossesor per block of size 1, where nothing is done. When done, we are left with one possible occurrence (or none) in each block of size $\sqrt{r}$, thus $\sqrt{r}$ possible occurrences altogether. Then in $O(1)$ time make all duels as described above. We are left with a

single possible occurrence (or none) in each block of size r.

The algorithm described above takes $O(\log\log m)$ time but is not optimal; it requires n processors. To achieve optimality we first partition our block into small blocks of size $\log\log r$. To each one of the $r/\log\log r$ small blocks assign a processor and make duels between pairs using a sequential algorithm till left with at most one possible occurrence in each small block. Then, proceed with the $O(\log\log r)$ algorithm having at most $r/\log\log r$ possible occurrences to start with. Since we have n/r blocks and in each block we used $r/\log\log r$ processors, we need a total of $n/\log\log r$ processors for this computation. Left with at most n/r possible occurrences, we can use the $O(1)$ algorithm we described in the introduction to verify these occurrences. The next step depends on the periodicity of the pattern, we have two cases:

1. The pattern is not periodic ($m < 2p$, $r = m/2$): Verify the whole pattern at each possible occurrences. This can be done using $\frac{mn}{r} = 2n$ processors in $O(1)$ time.

2. The pattern is periodic:

   - Verify at each possible occurrence in the text only the first $2p$ symbols of the pattern. This can be done using only $2n$ processors in $O(1)$ time.
   - **Find the initial** occurrence and the corresponding final occurrence: First find all initial occurrences and final occurrences. Then, find the maximal initial occurrence in the first $m-p$ symbols and the corresponding final occurrence. This can be done in

$O(1)$ time using m processors on our weak CRCW-PRAM (see section 7).

- Verify v right after the final occurrence. Note that v occurs after each nonfinal occurrence since v is a prefix of u.
- For each verified occurrence of u^2 check if enough occurrences follow and if followed by a verified occurrence of v. This can be done using the position of the initial occurrence and the final occurrence, and the information about v computed in the previous step.

Both 1 and 2 can be done in $O(1)$ time using n processors or $O(\log\log m)$ time using $n/\log\log m$ processors.

6 Processing the pattern

The $WITNESS$ array which we used in the text processing stage is computed incrementally. Knowing that some witnesses are already computed in previous stages, one can compute more witnesses easily. Let i and j be two indices in the pattern such that $i < j < \lceil m/2 \rceil + 1$. If $s = WITNESS(j-i+1)$ is already computed then we can find at least one of $WITNESS(i)$ or $WITNESS(j)$ using a duel on the pattern as follows:

- If $s+i-1 \leq m$ then $s+i-1$ is also a witness either for i or for j.
- If $s+i-1 > m$ then either s is a witness for j or $s-j+i$ is a witness for i (see figure 2).

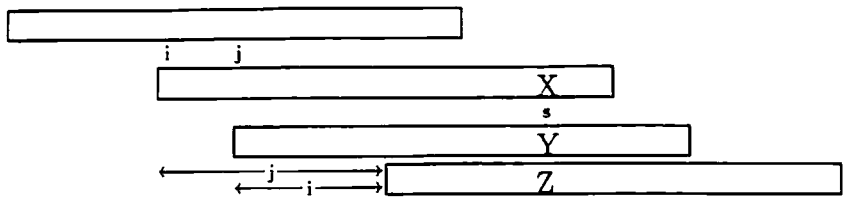

Figure 2. $X \neq Y$ and therefore we cannot have $Z = X$ and $Z = Y$.

First we describe an $O(\log\log m)$ non optimal algorithm. It works in stages and it has at most $\log\log m$ stages. Let $k_i = m^{1-2^{-i}}$, $k_0 = 1$. At the end of stage i, we have at most one uncomputed witness in each block of size k_i. The only uncomputed index in the first block is 1.

1. At the beginning of stage i we have at most k_i/k_{i-1} uncomputed witnesses in the first k_i–block. Try to compute them using the naive algorithm on $PATTERN(1 \cdots 2k_i)$ only. This takes $O(1)$ time using $2k_i \frac{k_i}{k_{i-1}} = 2m$ processors.

2. If we succeed in producing witnesses for all the indices in the first block (all but the first for which there is no witness), compute witnesses in each following block of the same size using the optimal duel algorithm described in the text processing section. This takes $O(\log\log m)$ time only for the first stage. In the following stages, we will have at most $\sqrt{m}$ indices for which we have no witness, and duels can be done in O(1) time.

3. If we fail to produce a witness for some $2 \leq j \leq k_i$, it follows that $PATTERN(1 \cdots 2k_i)$ is periodic with period length p, where $p = j - 1$ and j is the smallest index of an uncomputed witness. By

the periodicity properties mentioned above, all uncomputed indices within the first block are of the form $kp+1$. Check periodicity with period length p to the end of the pattern. If p turns out to be the length of the period of the pattern, the pattern analysis is done and we can proceed with the text analysis. Otherwise, the smallest witness found is good also for all the indices of the form $kp+1$ which are in the first k_i–block, and we can proceed with the duels as in 2.

These three steps seem to require simultaneous write of different values. In the next section we show that our weaker CRCW-PRAM can do it too. In order to make our algorithm optimal, we take a more careful look at the algorithm described above. We redefine our block sizes k_i as follows,

$$k_0 = 1$$

$$k_i = \frac{m^{1-2^{-i}}}{\log\log m}, \text{ for } i = 1 \cdots \log\log m$$

$$k_i = 2k_{i-1}, \text{ for } i > \log\log m,$$

introducing $\log\log\log m$ more stages. Using this new sequence, $m/\log\log m$ processors are enough for step 1 of the original algorithm. Step 2 will now take $\log\log m$ time for the first two stages after which we will have less than $\sqrt{\frac{m}{\log\log m}}$ uncomputed witnesses. However, step 3 still needs m processors and we need to modify the entire algorithm.

We have two kinds of stages: nonperiodic stages and periodic stages. Each kind is associated with certain initial conditions. The first stage is a nonperiodic stage 1 for which the initial conditions hold vacuously because $k_0 = 1$ and no witnesses are computed.

A nonperiodic stage i starts with at most one uncomputed witness in each k_{i-1}-block (in the first k_{i-1}-block the uncomputed witness is always the first). Moreover, all computed witnesses satisfy

$$WITNESS(l) \leq l + k_{i+1}. \qquad (1)$$

A periodic stage i starts with some uncomputed witnesses in the first k_{i-1}-block. They are all the indices of the form $kp + 1$, where p is the period length of the first k_i-block. In a periodic stage i all computed witnesses satisfy

$$WITNESS(l) \leq l + k_i \qquad (2)$$

and also,

$$WITNESS(l) \leq 2p \leq k_i \text{ for } 2 \leq l \leq p. \qquad (3)$$

In a nonperiodic stage i we execute step 1 of our original algorithm and if all witnesses in the first k_i-block are computed we perform the duels of step 2, which result in at most one uncomputed witness in any k_i-block. The new witnesses in the first k_i-block obviously satisfy $WITNESS(l) \leq 2k_i \leq k_{i+1}$. Hence, the new witnesses in the other k_i-blocks satisfy $WITNESS(l) < l + k_{i+2}$. So all computed witnesses satisfy (1) with i increased by 1. If all witnesses in the first k_i-block have been computed we proceed in a nonperiodic stage $i + 1$; otherwise, we verify p to be the period length of the first k_{i+1}-block. If it is not, we found the same witness ($\leq k_{i+1}$) for all the indices of the form $kp + 1$ in the first k_i-block and we continue with the duels of step 2 as in the previous case; otherwise we proceed with a periodic stage $i + 1$. In both cases, the initial conditions obviously hold.

In a periodic stage i we first check if p is the period length of the first k_{i+1}-block. In case it is, we use the periodicity to compute witnesses

for all indices l where $l \neq 1 \ (mod\, p)$ in the first k_i-block as follows. Let $j = \lfloor \frac{l-1}{p} \rfloor p$. Set $WITNESS(l) = j + WITNESS(l-j) \leq 2k_i \leq k_{i+1}$ (by (3)). We then proceed with a periodic stage $i+1$, and the initial conditions obviously hold. Actually, (3) might not hold immediately. By (2) we have $WITNESS(l) < k_{i+1}$ for $2 \leq l \leq p$. Since p is the period length of the first k_{i+1}-block, we can modify the witnesses to satisfy (3) as in section 3.

If we find that p is not the period length of the first k_{i+1}-block, we actually find at once a witness for all indices of the form $kp+1$ in the first k_{i-1}-block. This witness is not larger than k_{i+1}. We then perform the duels in each of the k_{i-1}-blocks, which result in all computed witnesses satisfying (1) and with at most one uncomputed witness in each k_{i-1}-block. These are the initial conditions for a nonperiodic stage i. We then proceed with a nonperiodic stage i. Note that unlike the nonoptimal algorithm, we perform duels only if the next stage is nonperiodic.

We now take a careful look at the last stage. Let r be maximal index such that $k_r < m$ and define $k_{r+1} = m$. As we have shown, duels can be made for all i and j where $i < j < \lceil m/2 \rceil + 1$, thus in a nonperiodic stage r everything works well if we perform duels only in the first half of the pattern. In a periodic stage r we either verify the period of the whole pattern, or we find a witness and enter a nonperiodic stage r.

Since we can be in a periodic stage i and a nonperiodic stage i at most once for each i, the total number of operations is $O(m)$ and by Brent's theorem our algorithm is optimal.

7 Some detail

Our computation model is a CRCW-PRAM where the only write conflict allowed is that processors can write the value 1 simultaneously into a memory location. The duels of our text analysis can obviously be implemented on such a model, while the duels of the pattern analysis and few other steps seem to require a stronger model of computation. We show how to implement the algorithm on our weaker model.

Consider the following problem: given an array of k integers, find the first 0. Fich, Ragde, and Wigderson [FRW] proposed the following $O(1)$ time algorithm using k processors on our weak CRCW-PRAM. Partition the array into $\sqrt{k}$ blocks of size $\sqrt{k}$. For each block find in $O(1)$ time if it has a 0 using $\sqrt{k}$ processors. Find the first block which has a 0 using $O(1)$ time minimum algorithm, and then find in that particular block the first position of a 0 using the same algorithm.

Using this algorithm, we find the initial occurrence, the final occurrence and witnesses in the first block in any stage of the pattern analysis without increasing our time/processor bounds on our weak CRCW-PRAM. The implemetation of finding the initial occurrence, the final occurrence and witnesses is obvious. However, the duels of the pattern analysis need to be done carefully. Suppose we perform duels among h indices, using h^2 processors. Each processor will write to a **different** memory location; then assign h processors to each of the h indices and check if a witness was found using the algorithm mentioned above.

We left out the details of the processor allocation for the duels since it can be done exactly as in Shiloach and Vishkin's [SV] maximum find-

ing algorithm. We need to calculate some sizes for our algorithm and for the usage of Brent's theorem (i.e. k_i's). $\lfloor \log \log m \rfloor$ can be calculated in $O(\log \log m)$ time using a single processor and square roots can be computed in $O(1)$ time on few processors as in [SV].

As in [G] the text analysis can also be done in $O(\log 1/\epsilon)$ time using nm^{ϵ} processors and the pattern analysis in $O(1/\epsilon)$ time using $m^{1+\epsilon}$ processors.

References

[BSV] Berkman, O., Schieber, B., and Vishkin, U. (1988), Some doubly logarithmic optimal parallel algorithms based on finding nearest smallers, *preprint.*

[BHa] Beame, P., and Hastad, J. (1987), Optimal Bound for Decision Problems on the CREW PRAM, *Proc. 19th ACM Symp. on Theory of Computing (1987)*, 83-93.

[BH] Borodin, A., and Hopcroft, J. E. (1985), Routing, merging, and sorting on parallel models of comparison, *J. of Comp. and System Sci. 30*, 130-145.

[BM] Boyer, R. S., and Moore, J. S. (1977), A fast string searching algorithm, *Comm. ACM 20*, 762-772.

[B] Brent, R. P. (1974), The parallel evaluation of general arithmetic expressions, *J. ACM 21*, 201-206.

[FRW] Fich, F. E., Ragde, R. L., and Wigderson, A. (1984), Relations between concurrent-write models of parallel computation, *Proc. 3rd ACM symp. on principles of distributed computing*, 179-189.

[G] Galil, Z. (1985), Optimal parallel algorithms for string matching, *Information and Control 67*, 144-157.

[KMP] Knuth, D. E., Morris, J. H. and Pratt, V. R. (1977), Fast pattern matching in strings, *SIAM J. comput. 6*, 322-350.

[Kr] **Kruskal**, C. P. (1983), Searching, merging, and sorting in parallel computation, *IEEE trans. on computers 32*, 942-946.

[LS] Lyndon, R. C., and Schutzenberger, M. P. (1962), The equation $a^M = b^N c^P$ in a free group, *Michigan Math. J. 9*, 289-298.

[Sc] Schieber, B. (1987), Design and analysis of some parallel algorithms, *Ph.D. Thesis, Tel-Aviv University.*

[ScV] Schieber, B., and Vishkin, U. (1987), The parallel complexity of finding all nearest neighbors in convex polygons, *preprint.*

[SV] Shiloach, Y. and Vishkin, U. (1981), Finding the maximum, merging and sorting in a parallel computation model, *J. Algorithms 2*, 88-102.

[Va] Valiant, L. G. (1975), Parallelism in comparison models, *SIAM J. of comput. 4*, 348-355.

[V] Vishkin, U. (1985), Optimal parallel pattern matching in strings, *Information and Control 67*, 91-113.